Pull Requests and Github

Pull Requests and Github

A practical guide to the workflow and best practices for submitting your assignments using GitHub.

Super Small Git Refresher

Git

- Distributed version control system
- Allows multiple devs to work on the same code

Pasted image 20250910132216.png

Merging

`git merge` allows you to integrate changes from different branches into a single branch. When you merge one branch into another, Git creates a new "merge commit" that combines the histories of both branches. This merge commit has two parent commits, preserving the individual development timelines.

There are two main types of merges:

- **Fast-Forward Merge**
- **Three-Way Merge:** When the branches have diverged, Git finds a common ancestor commit and creates a new merge commit that combines the changes from both branches since that common ancestor.

Pasted image 20250910134520.png

Rebasing

`git rebase` is a command that rewrites the commit history by reapplying your commits on top of another branch. Essentially, it changes the base of your branch from one commit to another.

Instead of creating a merge commit, rebasing takes the commits from your current branch and replays them, one by one, on top of the latest commit of the target branch. This results in a linear and cleaner commit history.

Pasted image 20250910160326.png

> & **Never rebase a branch that has been shared with others and that they are actively working on.** People will hate you, and nobody likes to be hated.

When To Use Each

When to use Merging

- When you are working on a shared or public branch where preserving the full history is important.
- In collaborative team environments to maintain a clear context of when and why branches were combined.
- For long-running feature branches to clearly show how they were integrated.

When to use Rebasing

- On your local or private branches to clean up your commit history before creating a pull request.
- When you want to maintain a linear and easy-to-read project history.
- To keep your feature branch up-to-date with the latest changes from the main branch.

For more info check out this article by Subodh Shetty and this post by No-Firefighter-6753 on reddit.

PRs

- PRs are the official way to submit homework
- Enable focused feedback and code review
- Create a clean, auditable history of your work

How to Submit HW

For HW1 (Getting Started):

1. **Fork the Template Repository:** Go to https://github.com/ivanearisty/oss-taapp or https://github.com/adithyab-20/ospsd-ta-task/pull/1 fork the repo.
2. **Clone Your Fork Locally:** Clone the repository from your account to your local machine.
3. **Create Your hw1 Branch:** The forked repository already has a root branch. Create your working branch from it.
4. **Complete your work** on the hw1 branch, then push it to your fork: git push -u origin hw1.
5. **Open a Pull Request on GitHub:** The **base** branch (the one you want to merge into) should be root, and the **compare** branch should be your hw1.

For HW2 and Beyond:

1. **Branch from your previous work.** Check out your hw1 branch and create your new hw2 branch from there (git checkout hw1; git checkout -b hw2).
2. **Complete your work** on the hw2 branch and push it.
3. **Open a Pull Request on GitHub:** The **base** branch must be hw1, and the **compare** branch should be hw2.

This ensures your PR only shows the new, incremental changes for HW2.

How to make edits and implement feedback

When you get feedback on a previous assignment (e.g., HW1) after you've already started the next one (HW2), follow this precise workflow:

1. **Switch to the older branch:** git checkout hw1
2. **Make the required changes,** commit them, and push the updates to the hw1 branch.
3. **Switch back to your current branch:** git checkout hw2
4. **Rebase your work:** Run git rebase hw1. This reapplies your HW2 changes on top of the now-fixed HW1 branch.
5. **Force-push your updated branch:** git push --force-with-lease origin hw2.

This keeps your history clean and integrates the corrections seamlessly into your latest work.

What not to do

- **Don't branch off of main:** Never target main or root for your HW2, HW3, etc. PRs. This clutters your PR with changes from all previous assignments, making it impossible to review effectively.
- **Don't make massive PRs:** Keep PRs ”minimal and relevant.” A pull request should contain work for a single, specific task.
- **Don't submit with failing CI:** The Continuous Integration (CI) checks must pass before your PR is considered ready for review.
- **Don't be a ”Shipped Buddy”:** Do not approve a teammate's PR without a proper review. The goal is to improve code quality, not just get your work merged.

How main should look like

At the end of the semester, your main (or root) branch should have a minimal and clean commit history.

The branching strategy we use (hw2 -> hw1, etc.) is designed to keep your development history separate from the main line. After all feedback is incorporated and your HW3 is finalized, you can perform a ”Squash and Merge” of your final work into main.

This can result in the main branch having just a single, clean commit representing the entire finished project. This is highly desirable in a template repository, as it ensures future users don't inherit a long and messy development history.

Final Tip: use lazygit